preguntas

RED

punto 2

código 1

punto 1

PROYECTO REDES

FUNDAMENTOS DE REDES

TONY SANTIAGO MONTES

JOHN ESTEBAN PULIDO SALINAS

preguntas

RED

punto 2

código 1

PUNTO 1

Algorithm 1

Tahoe

Aplicacion al proyecto

Algorithm 2

Reno

Optimización respecto a Tahoe

Gráficas 3

Final

Tahoe vs Reno Gráficas

punto 1

preguntas

RED

punto 2

Carga de datos

Csv a matríz

Se leen los datos del csv a una matríz de listas

Tahoe código

Reno

Código de optimización tipo tahoe

Reno código

Tahoe

Código de optimización tipo reno

código 1

punto 1

PUNTO 2

Algorithm 1

Procedimiento General

Carga de Tablas

Algorithm 2

Procedimiento General

Cálculo Longitud de Cola

Gráficas 3

Tiempo vs Tiempo en Cola

preguntas

Algorithm 1

Implementación AQM RED

Random Early Detection

Gráficas 2

Tiempo vs Tiempo en Cola – con AQM

RED

punto 2

código 1

punto 1

Revisar

- Del primer punto revisar la diapositiva 2 (“Código 1”). La carga de archivos, la impementación tahoe y la implementación reno
- Del segundo punto revisar procedimiento general: Tablas y Cálculo de la Longitud de Cola.

IMPORTANTE

Para poder acceder a las diapostivas con la información dar click o ctrl +enter en los botones:

Para ir a la diapositiva o para regresar a cada una respectivamente

Drive

Link con los archivos .py con los que desarrollamos el proyecto: https://drive.google.com/drive/folders/1nXEeZPCLOO7WpG3wHIgdlKpB7UaIQhhd?usp=sharing

Algorithm 1

Tahoe

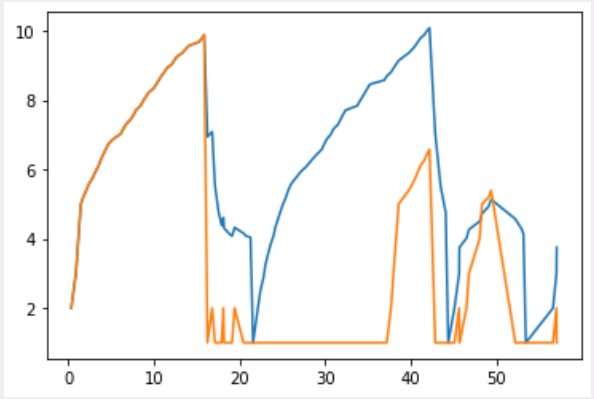
```
ventana = max valor de congestion
ssthresh = ventana/2
for paquete in paquetes:
        menor = congestion < ventana
        if paquete[ack] not exist and congestion <  ssthresh and menor:
                congestion = congestion + 1
        elif paquete[ack] not exist and congestion >=  ssthresh and menor:
                congestion = congestion + (1/congestion)
        elif paquete[ack] exist and menor :
                congestion = 1
        elif not menor:
                pass
```

Algorithm 2

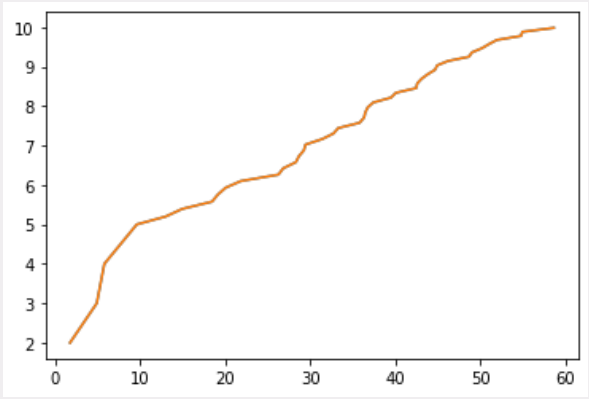
Reno

```
ventana = max valor de congestion
ssthresh = ventana/2
for paquete in paquetes:
        menor = congestion < ventana
        if paquete[ack] not exist and congestion <  ssthresh and menor:
                congestion = congestion + 1
        elif paquete[ack] not exist and congestion >=  ssthresh and menor:
                congestion = congestion + (1/congestion)
        elif paquete[ack] exist and repetidos < 3 and menor:
                repetidos[ack] += 1
                ssthrersh = congestion/2
                congestion = sstresh +3
        elif paquete[ack] exist and repetidos == 3:
                congestion = 1
                repetidos[ack] += 1
        elif paquete[ack] exist and repetidos > 3 and menor:
                congestion = congestion + (1/congestion)
                repetidos[ack] += 1
        elif not menor:
                pass
```

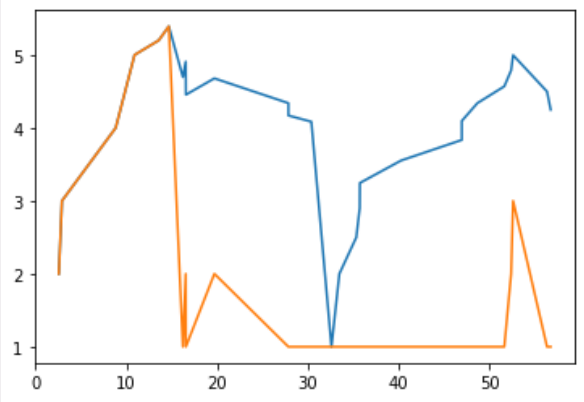
Gráficas



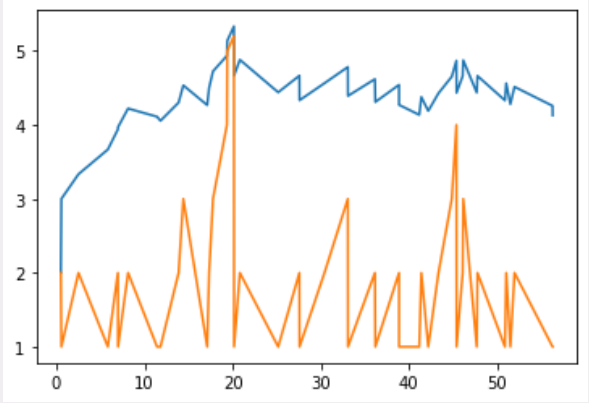
from IP(“81.131.67.131”) to IP(“210.146.64.4”)



from IP(“81.131.67.131”) to IP(“211.28.6.30”)



from IP(“81.131.67.131”) to IP(“213.19.160.190”)



from IP(“81.131.67.131”) to IP(“38.115.4.204”)

Código

Carga de datos

```
def cargar(datos):
    # Se cargan los datos
    matriz = [] #Matríz de datos
    archivo = open(datos,'r') #Se abre el archivo
    linea = archivo.readline().replace('\n','')#Se lee la primera linea
    linea = archivo.readline().replace('\n','')#Se lee la primera linea
    while len(linea) > 0:
        linea = linea.split(',')
        if linea[2] == '"81.131.67.131"' and linea[3] == '"211.28.6.30"':
            matriz.append(linea)
        linea = archivo.readline().replace('\n','')#Se vuelve a leer la linea
    archivo.close()
    return matriz
```

Código

Tahoe

```
def tahoe(matriz):
    congestiones =[]
    tiempos = []
    acks = []
    congestion = 1
    ventana = 10
    ssthresh = 5
    for i in range(len(matriz)):
        ack = matriz[i][7]
        rtt = float(matriz[i][1])
        if ack == 0:
            acks = []
        if ack not in acks and congestion < ssthresh and congestion < ventana:
            congestion = congestion + 1
            congestiones.append(congestion)
        elif ack not in acks and congestion >= ssthresh  and congestion < ventana:
            congestion = congestion + (1/congestion)
            congestiones.append(congestion)
        elif ack in acks:
            congestion = 1
            congestiones.append(congestion)
        elif congestion >= ventana:
            congestiones.append(congestion)
        tiempos.append(rtt)
        acks.append(ack)
    plt.plot(tiempos, congestiones)
```

Código

Reno

```
def reno(matriz):
    congestiones =[]
    tiempos = []
    acks = []
    num = []
    congestion = 1
    ventana = 10
    ssthresh = 5
    repetidos = 0
    contador = 0
    for i in range(len(matriz)):
        ack = matriz[i][7]
        rtt = float(matriz[i][1])
        if ack == 0:
            acks = []
        if ack not in acks and congestion < ssthresh and congestion < ventana:
            congestion = congestion + 1
            congestiones.append(congestion)
            repetidos = 0
        elif ack not in acks and congestion >= ssthresh  and congestion < ventana:
            congestion = congestion + (1/congestion)
            congestiones.append(congestion)
            repetidos = 0
        elif ack in acks and repetidos == 3:
            congestion = 1
            repetidos += 1
            congestiones.append(congestion)
        elif ack in acks and repetidos > 3 and congestion < ventana:
            congestion = congestion + (1/congestion)
            repetidos += 1
            congestiones.append(congestion)
        elif ack in acks and repetidos < 3:
            repetidos += 1
            ssthresh = congestion/2
            congestion = ssthresh + 3
            congestiones.append(congestion)
        elif congestion >= ventana:
            congestiones.append(congestion)
        contador += 1
        num.append(contador)
        tiempos.append(rtt)
        acks.append(ack)
    plt.plot(tiempos, congestiones)
```

Algorithm

Tablas

```
tabla = [[], [], [], []]
mu = lbd/rho
for fila in range(len(matriz)):  # Se recorren las filas de la matríz
    # Se calcula el tiempo de transmisión total
    tt = matriz[fila][1]/mu
    if fila == 0:  # Se analiza si es el primer dato
        # Se agrega el tiempo de inicio
        tabla[1].append(matriz[fila][0])
        tabla[3].append(tt)  # Tiempo en que termina la transmisión
    # Si el fin de transmisión
    elif tabla[3][fila-1] > matriz[fila][0]:
        tabla[1].append(tabla[3][fila-1])
        tabla[3].append(tt+tabla[3][fila-1])
    else:
        tabla[1].append(matriz[fila][0])
        tabla[3].append(matriz[fila][0]+tt)

    tabla[0].append(matriz[fila][0])
    tabla[2].append(tt)
```

Algorithm

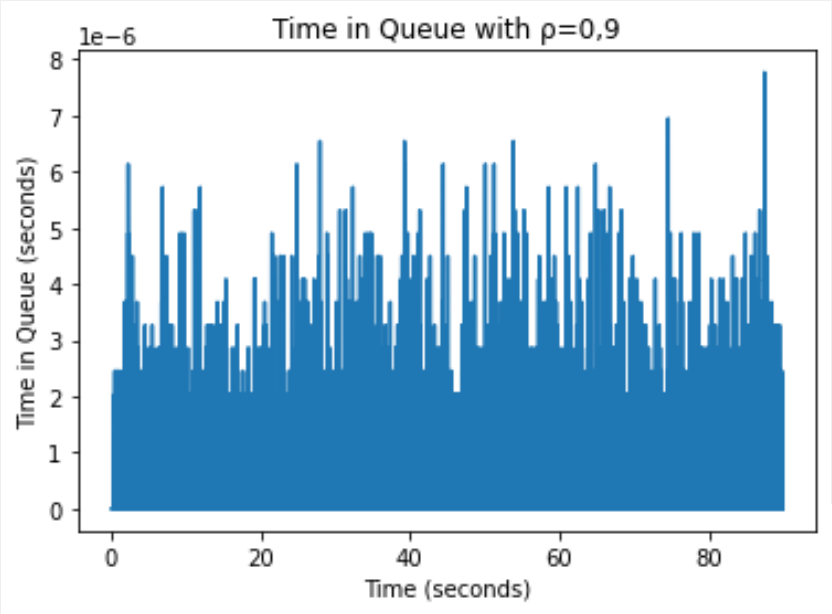
Longitud de Cola

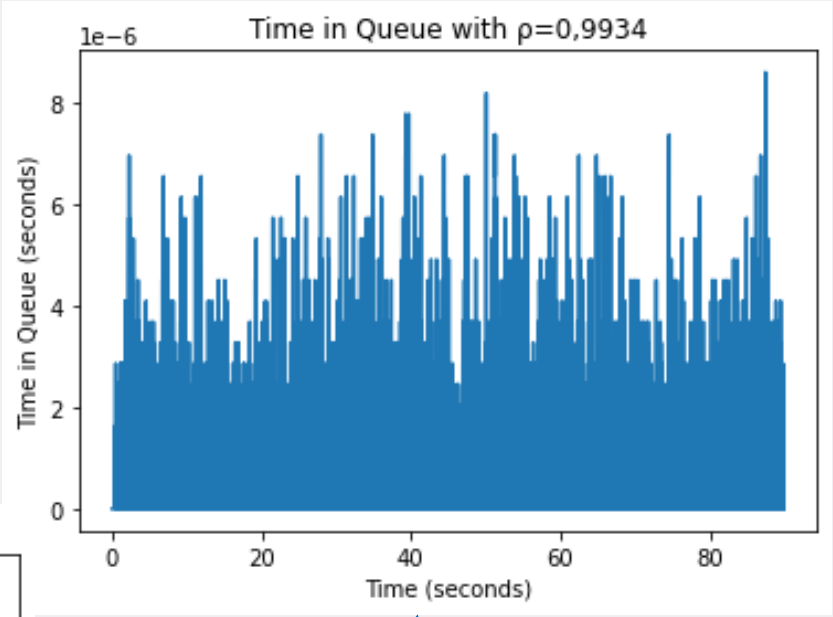
```
"""Paso 3. Calcular lista de 0,1,-1 por paquete"""
lst = []
for i in range(len(tabla[0])):
    if tabla[0][i] == tabla[1][i]:
        lst.append((tabla[0][i], 0, matriz[i][1]))
    else:
        lst.append((tabla[0][i],  1, matriz[i][1]))
        lst.append((tabla[1][i], -1, matriz[i][1]))

lst.sort(key=lambda x: x[0])
```

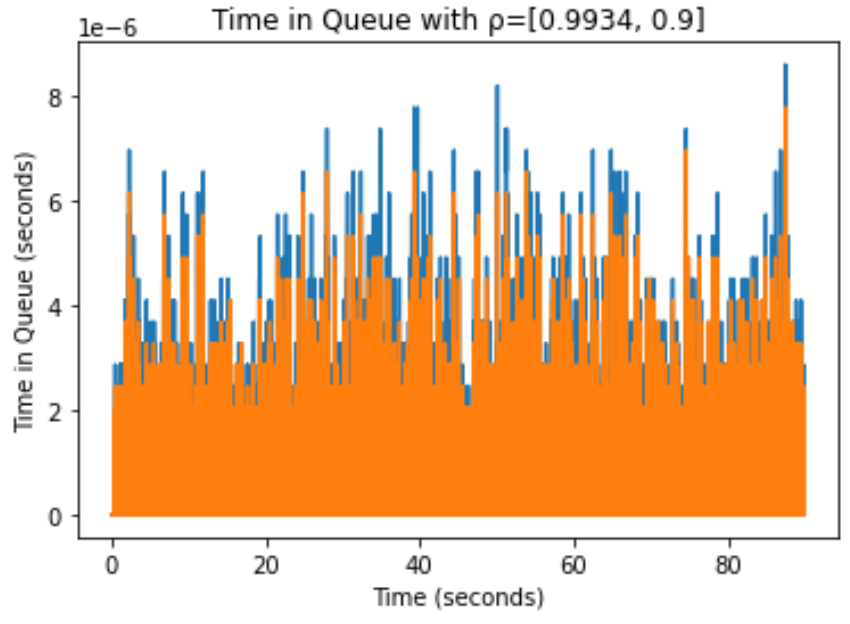
```
"""Paso 5. Calcular Qlen en cada toma"""
Wlen = []
times = []
cont = 0
for i in range(len(lst)):
    cont += lst[i][1]  # Suma 0, 1 o -1
    times.append(lst[i][0])
    Wlen.append(cont/lbd) # cont = Qlen
```

Gráficas

Sin aplicar AQM







Algorithm

Random Early Detection

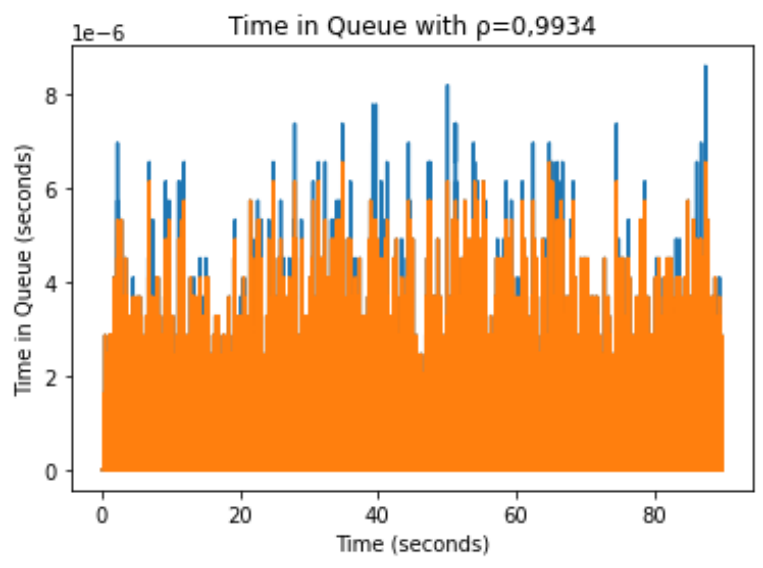
```python
def RED(lst, minT, thresh, maxProb) -> list:
            mtx = [[lst[0][0], lst[0][2]]]
            counter = lst[0][1]
            total = []
            avgQlen = 0
            numdiscarted = 0
            for i in range(1,len(lst)):
                counter += lst[i][1]
                if len(total) < thresh:
                    total.append(counter)
                else:
                    total.append(counter)
                    del total[0]
                avgQlen = sum(total)/thresh
                if lst[i][1] != -1:
                    # Cuando es -1 el tiempo (es de llegada) no se tiene en cuenta para la matr
                    if avgQlen < minT:
                        mtx.append([lst[i][0], lst[i][2]]) #tiempo, bytes
                    elif avgQlen > minT + thresh:
                        pass # Se descarta
                        numdiscarted += 1
                    else:
                        probability = maxProb*(1 - (minT + thresh - avgQlen)/thresh) # Fórmula
                        num = randint(1, 100)/100 # número aleatorio entre 0.01 y 1
                        if num < probability:
                            pass # Se descarta
                            numdiscarted += 1
                        else:
                            mtx.append([lst[i][0], lst[i][2]])
            print(f"RED -> Elements discarted = {numdiscarted}")
            return mtx
```

Con RED

Sin RED

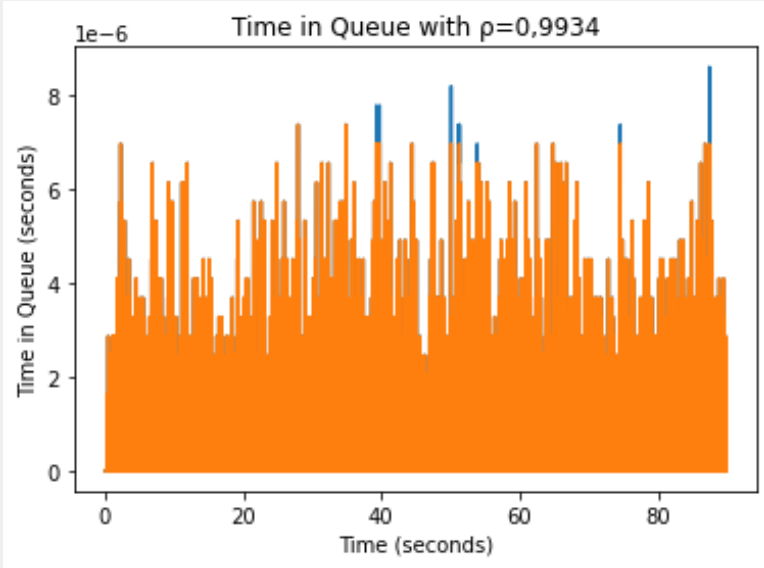
Gráficas
Aplicando AQM=RED

maxProbability



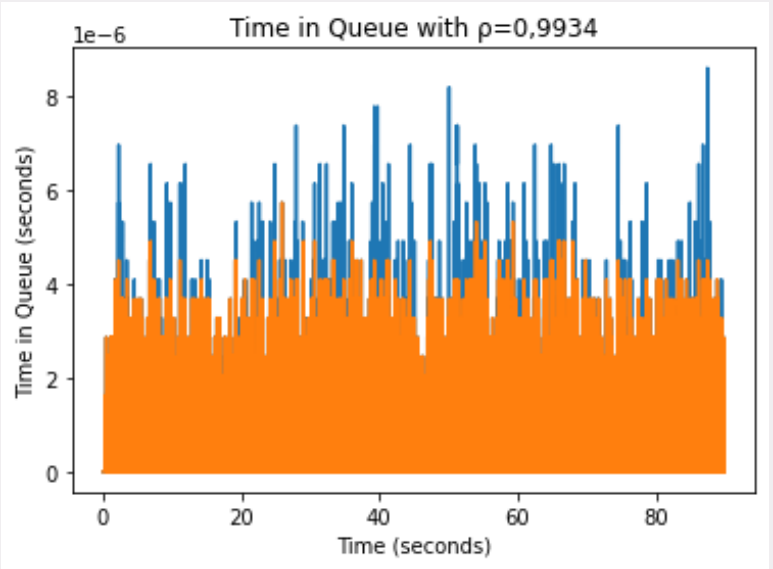
Parámetros:
- minThresh = 5
- threshold = 10
- maxProbability = 25%

Paquetes Descartados: 553



Parámetros:
- minThresh = 5
- threshold = 10
- maxProbability = 1%

Paquetes Descartados: 110



Parámetros:
- minThresh = 5
- threshold = 10
- maxProbability = 100%

Paquetes Descartados: 1917

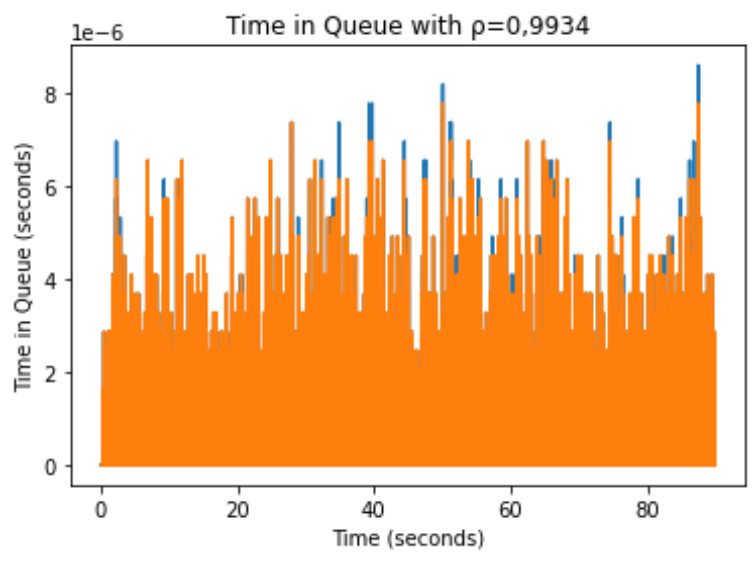
Con RED

Sin RED

Gráficas 2

Aplicando AQM=RED

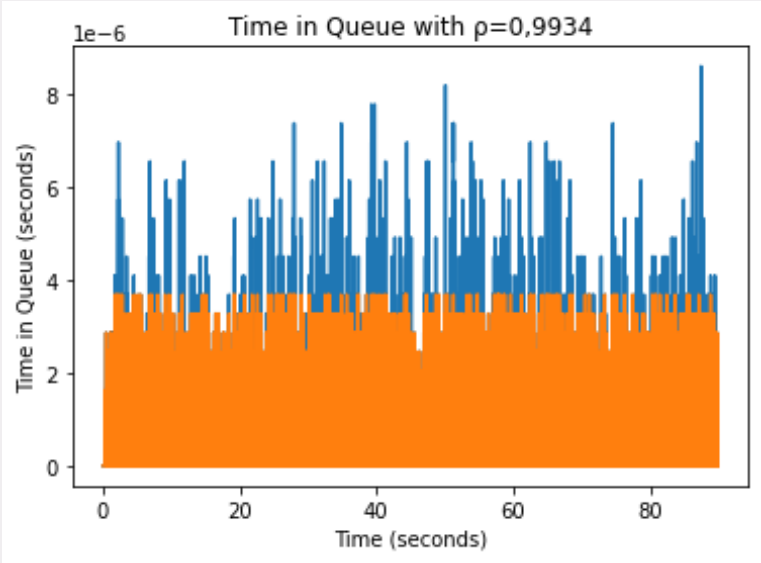
threshold



Parámetros:

- minThresh = 5
- threshold = 20
- maxProbability = 25%

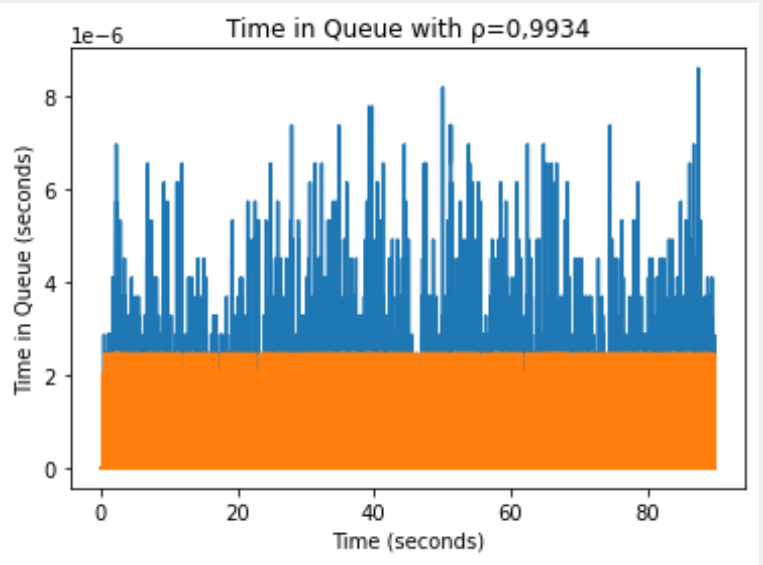
Paquetes Descartados: 188



Parámetros:

- minThresh = 5
- threshold = 3
- maxProbability = 25%

Paquetes Descartados: 3181



Parámetros:

- minThresh = 5
- threshold = 1
- maxProbability = 25%

Paquetes Descartados: 6528

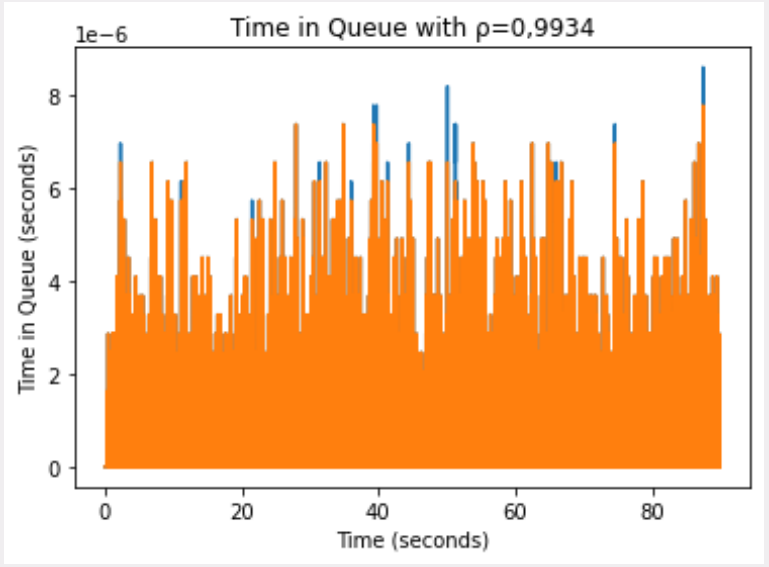
Con RED

Sin RED

Gráficas 3

Aplicando AQM=RED

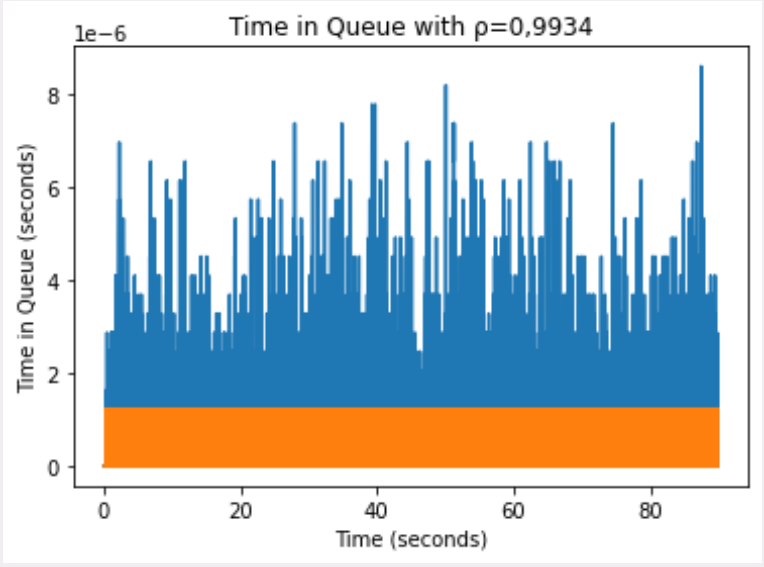
minThresh



Parámetros:

- minThresh = 10
- threshold = 10
- maxProbability = 25%

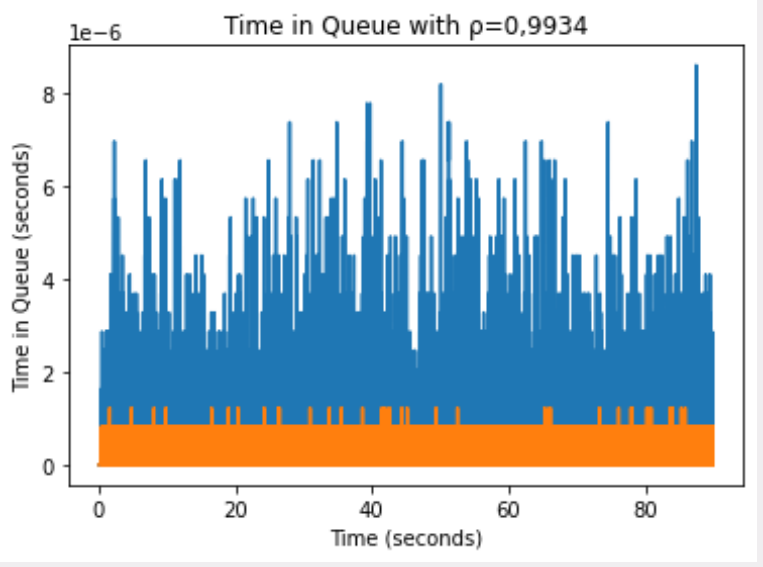
Paquetes Descartados: 63



Parámetros:

- minThresh = 2
- threshold = 1
- maxProbability = 1%

Paquetes Descartados: 17797



Parámetros:

- minThresh = 2
- threshold = 1
- maxProbability = 100%

Paquetes Descartados: 28028